

Inter-Paradigm Translation of Process Models using Simulation and Mining

Lars Ackermann¹, Stefan Schöning² and Stefan Jablonski¹

¹ University of Bayreuth, Germany
{firstname.surname}@uni-bayreuth.de

² Vienna University of Economics and Business, Austria

Abstract Process modeling is usually done using imperative modeling languages like BPMN or EPCs. In order to cope with the complexity of human-centric and flexible business processes several declarative process modeling languages (DPMLs) have been developed during the last years. DPMLs allow for the specification of constraints that restrict execution flows. They differ widely in terms of their level of expressiveness and tool support. Furthermore, research has shown that the understandability of declarative process models is rather low. Since there are applications for both classes of process modeling languages, there arises a need for an automatic translation of process models from one language into another. Our approach is based upon well-established methodologies in process management for process model simulation and process mining without requiring the specification of model transformation rules. In this paper, we present the technique in principle and evaluate it by transforming process models between two exemplary process modeling languages.

Keywords: process model translation, simulation, process mining

1 Introduction

Two different types of processes can be distinguished [1]: well-structured routine processes with exactly predescribed control flow and flexible processes whose control flow evolves at run time without being fully predefined a priori. In a similar way, two different representational paradigms can be distinguished: imperative process models like BPMN³ models describe which activities can be executed next and declarative models define execution constraints that the process has to satisfy. The more constraints we add to the model, the less eligible execution alternatives remain. As flexible processes may not be completely known a priori, they can often be captured more easily using a declarative rather than an imperative modelling approach [2–4]. Due to the rapidly increasing interest several declarative languages like *Declare* [5], *Dynamic Condition Response* (DCR) Graphs [6] or *Declarative Process Intermediate Language* (DPIL) [7] have been developed in parallel and can be used to represent these models. Consequently, flexible processes in organizations are frequently modeled in several different notations. Due to several reasons in many cases a translation of process models to a

³ The BPMN 2.0 standard is available at <http://www.omg.org/spec/BPMN/2.0/>.

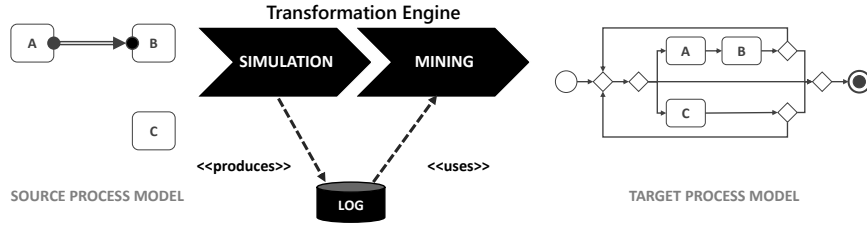


Fig. 1: Overview of the model transformation approach

different language is desired: *(i)* since declarative languages are difficult to learn and understand [3], users and analysts prefer the representation of a process in an imperative notation, *(ii)* even if the user is familiar with a particular notation neither imperative nor declarative languages are superior for all use cases [8], *(iii)* adopted process execution systems as well as analysis tools are tailored to a specific language and *(iv)* since process modeling is an iterative task, the most appropriate representation for the evolving process model may switch from a declarative to an imperative nature and vice versa. To facilitate these scenarios, a cross-paradigm process model transformation technique is needed. While contemporary research mainly focuses on transforming process models between different imperative modeling languages, approaches that comprise declarative languages are still rare [8].

We fill this research gap by introducing a two-phase, bi-directional process model transformation approach that is based upon existing process simulation and mining techniques. The principle is highlighted in Fig. 1. Model-to-model transformation techniques usually involve the creation of transformation rules which is an error-prone and cumbersome task [9]. Hence, we created an approach that does not require the definition of transformation rules. First, a set of valid execution traces of the process is automatically generated by simulating the source model. Second, the resulting event log is analyzed with a process mining approach that uses the target language to represent the discovered model. For the work at hand we use Declare as a representative of declarative languages and BPMN as the imperative language. However, note that the approach works with every language framework that provides model simulation and mining functionality. We evaluate functionality and performance by transforming four simplistic examples as well as two real-life process models between BPMN and Declare.

The remainder of this paper is structured as follows: Section 2 describes the fundamentals of declarative process modeling at the example of Declare as well declarative and imperative simulation and mining. In Section 4 we introduce our approach to transform declarative process models. The approach is evaluated in Section 5. We discuss related work in Sec. 6 and Section 7 concludes the paper.

2 Background and Preliminaries

In this section we introduce declarative process modeling as well as the simulation and mining of declarative process models.

2.1 Declarative Process Modeling

Research has shown that DPMLs are able to cope with a high degree of flexibility [10]. The basic idea is that, without modeling anything, everything is allowed. To restrict this maximum flexibility, DPMLs like Declare allow for formulating rules, i.e., constraints which form a forbidden region. An example is given with the single constraint *ChainSuccession(A, B)* in Fig.1, which means that task B must be executed directly after performing task A. Task C can be performed anytime. The corresponding BPMN model mainly consists of a combination of exclusive gateways. Declare focuses almost completely on control flow and, thus equivalent BPMN models may only consist of control flow elements as well. A brief discussion of issues related to differences in the expressiveness of the two languages is given in Sec. 4.1. Declarative and imperative models are in a manner opposed. If one adds an additional constraint to a declarative model, this usually results in removing elements in the imperative model and vice versa. If, for instance, we add the two constraints *Existence(A)* and *Existence(C)* to the source process model in Fig. 1, the edge directly leading to the process termination event must be removed. For a transformation approach this means that the identification of appropriate transformation rules would be even more complicated, because a control-flow element in the source language does not necessarily relate to the same set of control-flow elements in the target language in all cases.

2.2 Process Simulation and Process Mining

In this section, we briefly describe the two methods our transformation approach is based on. Simulating process models is well-known as a cost-reducing alternative to analyzing real-world behavior and properties of business processes [11]. Though, there are different simulation types, for our purpose, we exclusively refer to the principle of *Discrete-event Simulation (DES)* [12]. DES is based upon the assumption that all relevant system state changes can be expressed using discrete sequences of events. By implication this means that there is no invisible state change between two immediately consecutive events. This assumption is valid since we use a simulation technique for the purpose of model translation. This means that, in our case, a source process model fully describes the universe of system state changes. For the application in our approach we use simulation techniques to generate exemplary snapshots of process executions allowed by an underlying process model. The produced simulation results are the already mentioned event logs, containing sets of exemplary process execution traces. These logs are then consumed by process mining techniques.

Process Mining aims at discovering processes by extracting knowledge from event logs, e.g., by generating a process model reflecting the behaviour recorded

in the logs [13]. There are plenty of process mining algorithms available that focus on discovering imperative process models, e.g., the simplistic *Alpha miner* [13] or the *Heuristics Miner* [14]. Recently, tools to discover declarative process models like DeclareMiner [15] or MINERful [16] have been developed as well. In the approach at hand, we use process mining techniques to automatically model the simulated behaviour in the chosen target language.

3 Challenges and Preconditions

Our approach at hand requires some prior analysis and raises some challenges we have to deal with. Probably the most important as well as the most trivial challenge is to prevent the transformation approach from causing information loss (*CP1*). This means that source and target model must be semantically equivalent. However, the approach at hand does not consider interpolation methods but instead assumes that source and target language have the same semantic expressiveness. We therefore provide a limited comparative analysis of the expressiveness of Declare and BPMN in section 4.1. (*CP2*) complements the issue of expressiveness. It must be examined whether a process log is expressive enough to be able to cover the behavioral semantics of a process model. While (*CP2*) discusses the general *ability* of log data to preserve the behavioral semantics of a process model, we now have to make sure that it *actually contains* the required execution traces [14]. Therefor both transformation steps, simulation as well as process mining, require appropriate parameterizations (*CP3*). Many process mining approaches suggest that the best parametrization is data-dependent and can therefore be determined in particular only. Hence, it is necessary to provide a strategy for the determination of well-fitting parameter values.

4 Contribution

The translation of a model specified in one language to another is usually done using *mapping rules*. In general, we have a set of n modeling languages that are able to describe the same domain. A translation system that uses this *direct*, mapping-rule-based translation principle has a complexity of $O(n(n-1)) = O(n^2)$ in terms of required rule sets n . Finding all rule sets for a system of modeling languages is, therefore, a time-consuming and cumbersome task [9]. On the contrary, our transformation approach is based on the following two core techniques: (i) Process Models Simulation and (ii) Process Mining. Therefore, our approach does not require the creation of transformation rules but uses the core idea to extract the *meaning* of a particular model by generating and analyzing valid instances of the model through simulation. Each simulation result is represented as an event log which is the usual input for process mining techniques such as [14, 15, 17]. This means that our transformation approach is based on the assumption that we are able to find appropriate simulation and mining technologies for the individual language pairs. In the case of our continuously used BPMN-Declare language pair several simulation and mining techniques are ready to use.

The following subsections briefly describe the principles and configuration strategy of these two core techniques. Furthermore, we briefly discuss issues related to the expressiveness of the particular modeling languages and of event log used as a transfer medium.

4.1 Language and Log Expressiveness

We have to discuss two key factors for our translation approach: (i) Differences in the expressiveness of the particular source and target language and (ii) potentially insufficient expressiveness of event logs.

Equal *Language Expressiveness* (CP1) means, in our context, that two languages, e.g. BPMN and Declare, are able to model the same semantics, no matter if the resulting model is imperative or declarative. Considering our two exemplary process modeling languages, we can easily find significant differences. Even though Declare is extendable, its expressiveness is currently limited to concepts that describe tasks and temporal or existence constraints. In contrast, BPMN allows for modeling of organizational associations as well as data flow and other elements. In order to provide a profound catalog that describes model patterns which can be translated successfully, an extensive comparison of the two particular process modeling languages is required. Due to the fact that such a deep analysis is currently not available and because this topic would go beyond the scope of this paper we choose example processes for our evaluation that can definitely be represented in both languages.

The second issue is the question of *Sufficient Log Expressiveness* (CP2). An event log “contains historical information about ‘When, How, and by Whome?’” [18]. An event log describes an example of a process execution and, hence, one possible trace through the source process model. Process mining techniques are built based upon the following assumptions regarding the log contents and structure: (i) a process consists of cases that are represented by traces, (ii) traces consist of events and (iii) events can have attributes like the activity name, a timestamp, associated resources and a transaction type [13]. An event can, therefore, unequivocally be associated with the corresponding activity, resources and the type of the system state change. All of these information describe a single state change but not dependencies between state changes. Thus, process mining techniques are limited to the information that can be encoded in sequential, discrete event logs. However, let us consider model (d) shown in Fig. 2. In order to extract this *chainPrecedence*(A, B) rule from a process event log, the following condition must be valid for all traces: Whenever an event occurs that refers to activity B then the event occurring immediately before⁴ must refer to the activity A. This suggests that temporal relationships can be extracted from the log if the latter’s quality and length is sufficient. However, the activity labeled with C in the same model is not restricted by any constraint. This means, by implication, that it can be executed arbitrarily often. Because a log has a finite length, we cannot encode this knowledge. Instead the mining technique could

⁴ Declare does not distinguish between different transaction types.

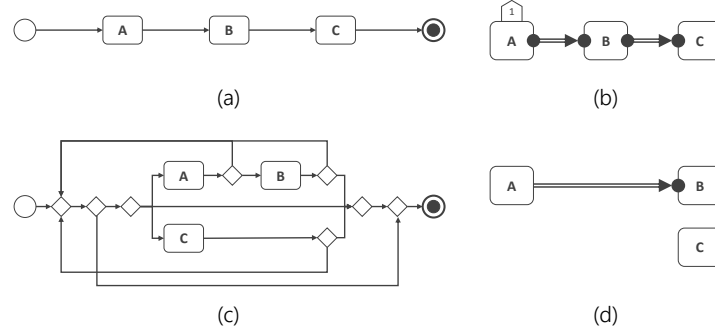


Fig. 2: Continuous example

use some threshold following the assumption, that, if a particular task has been executed n times, the number of possible executions is theoretically unlimited.

Like in the case of language expressiveness, a much deeper dive into the limitations of information encoding in discrete-event logs is required but would go beyond the scope of this paper. So far we briefly discussed, what information an event log is *able* to provide. The following three subsections focus on *if* and *how* we can make sure that the desired information are contained in the log.

4.2 General Simulation Parameters

There are two general properties which influence the transformation quality as well as the performance of the whole approach, i.e. (i) *the Number of Traces* (N) and (ii) *the Maximum Trace Length* (L).

Setting the value for N appropriately is the basis for including all relevant paths in the log. Considering the example BPMN model in Fig. 2 (c), there are several gateways whereby each unique decision leads to a new unique execution trace. Hence, we need a strategy for determining the minimum number of traces to include in the log. However, this number depends on the second parameter L . Without providing a value for L the simulation of a process model that allows for loops could hypothetically produce traces of infinite length. Thus, the potential number of different traces is also infinite. We therefore need an upper bound for L . The lower bound is governed by the process model itself.

The appropriate setting (*CP3*) of the introduced parameters depend on the source process model. In the case of the BPMN model in 2(a) the trace $\langle ABC \rangle$ describes the model's behavioral semantics exhaustively. Obviously this single trace does not represent the semantics of 2(c) appropriately, because of several decision points and loops. A simple formula to calculate the minimum number is shown in Eq. 1. This formula considers the size of the set of tasks ($|T|$) and is further based on the knowledge that the length of the longest possible sequence of tasks without repetition is given by L . The formula also factors in arbitrary

task orderings (the i th power) and shorter paths (the sum). Using this formula we do not need any information about the structure of the process model.

$$N \geq \sum_{i=0}^{L-1} |T|^i \quad (1) \quad L \geq 2|T| \quad (2)$$

Assuming that the simulation engine is able to produce shorter traces, too, the formula for L is based on the idea that the longest trace without repetition ($|T|$) could be executed at least twice (Eq. 2).

Both formulae describe just the absolute lower bound for both dimensions. We therefore suggested to choose both parameters significantly higher. However, since N increases exponentially with L , using this formula becomes expensive very fast. In practice considerably less traces are necessary, because the formula only considers tasks in arbitrary combinations without following the rules in the process model. Hence, our evaluation has a the twofold purpose to test our approach in general and to serve as a guideline for checking the quality of the transformation for a particular configuration in practice.

In order to increase the probability that all relevant traces are contained in the log, we have to ensure that all of them are generated with equally often. However, since this cannot be configured directly in the chosen tools, we only provide a simplistic configuration, which is discussed in the next subsection.

4.3 Simulating Imperative Process Models

The simulation technique has to be able to produce a certain number of traces of appropriate length. In contrast, simulation tools built for measuring KPIs usually use different configurable parameters [18,19]: *(i)* the Case-Arrival Process (*CAP*), *(ii)* the service times (*st*) for tasks as well as *(iii)* the probabilities for choices. Since our intent is to reuse existing techniques and technologies we have to map our desired simulation parameters from Sec. 2.2 to the implemented parameters of the particular simulation technique.

The *CAP* influences the number of traces that can be generated. In order to ensure that the desired amount of traces N is generated, the inter-arrival time (t_a) must be set to a constant value. Finally the minimum simulation duration d can be calculated according to the formula $d = \frac{N}{t_a}$. To be able to create a target model with equivalent semantics, we have to make sure that the behavioral semantics of the source model is implicitly described by the log data as accurately as possible. Given a BPMN model this means that all control-flow edges are used in the simulation step.

Another influencing factor is the task *service time*, i.e. the usual duration. For our purposes these service times have to be equal and constant for all tasks. Executing two tasks A and B in parallel with $st_B > st_A$ would always produce the same sequence during simulation: $\langle \dots AB \dots \rangle$. The subsequent Declare mining algorithm would falsely introduce a *chainsuccession*(A, B) instead of a correct *coexistence*(A, B) rule. With constant and equal values the ordering is completely random which actually is one intuition of a parallel gateway.

Probability distributions are used to simulate human decisions [18] at modeled gateways, which means that the outgoing edges are chosen according to a probability that follows this distribution. The probabilities for all outgoing edges of one gateway must sum up to one and, thus, the uniform-distributed probability can be calculated according to the formula $\frac{1}{n_{O,G}}$ with $n_{O,G}$ denoting the number of outgoing edges for gateway G . However, determining these probabilities only locally leads to significantly lower probabilities for traces on highly branched tasks. However, since we assume a completely unstructured process when developing Form. 1, in many cases we will generate far too much traces. Thus, we suggest this as an initial solution which is proved in our evaluation.

Configuring the maximum trace length L is slightly more complicated. The reason is that imperative processes are very strict in terms of valid endings of instances. This involves some kind of *look ahead* mechanism which is able to check whether the currently chosen step for a trace does still allow for finishing the whole process validly. Our approach restricts the trace length in an post-processing step based on a simulation of arbitrary length which is only restricted by the simulation duration. Afterwards we select only those traces which do not exceed the configured maximum trace length.

4.4 Simulating Declarative Process Models

The main difference between imperative and declarative process modeling languages is that the former means modeling allowed paths through the process explicitly utilizing directed-graph representations while the latter means modeling them implicitly based on rules. In [20] the authors presented an approach for simulating Declare models based on a six-step transformation technique. First, each activity name is mapped to one alphabetic character. Afterwards, the Declare model is transformed into a set of regular expressions. For each regular expression there exists an equivalent *Finite State Automaton (FSA)* which is derived in the third step. Each regular expression and, therefore, each FSA corresponds to one Declare constraint. To make sure that the produced traces respect all constraints the product of all automaton is calculated in step four. During the next step, the traces are generated by choosing a random path along the FSA product and by concatenating the characters for all passed transitions. In the sixth and last step the characters are mapped to the original activity names and the traces are written to a log file. Similar to the simulation of imperative process models, it is necessary to configure the parameters N and L . In [20] both parameters can be configured directly. In contrast, we have no influence on the probability distribution for the traces since the algorithm internally assumes a uniform distribution that assigns equal probabilities to all outgoing edges of each state in the FSA. Hence, again, there is a mismatch regarding the probability for highly branched paths as in the simulation for imperative models.

| Param | Value | Param | Value |
|--------------------------|-------|--------------------|-------|
| Relative-to-best | 0.0 | Ignore Event Types | false |
| Dependency | 49.0 | Minimum Support | 100.0 |
| Length-one/two loop | 0.0 | Alpha | 0.0 |
| Long distance | 100.0 | | |
| All tasks connected | true | | |
| Long distance dep. | true | | |
| Ignore loop dep. thresh. | false | | |

Table 1: Miner configurations: FHM (l), DMM (r)

4.5 Mining Imperative BPMN Process Models

In order to complete our tool chain we have to choose a mining technique and an appropriate configuration. We selected the Flexible Heuristics Miner (FHM) [14]. Though this mining algorithm first produces a so called *Causal Net* that must be later converted to BPMN, the advantages outweigh the disadvantages: (i) The algorithm is able to overcome the drawbacks of simpler approaches, such as the Alpha algorithm, (ii) it is specialized for dealing with complex constructs. This is very important since a “good” Declare model, which means the tasks and not too many constraints, would usually lead to a comparatively complex BPMN model. (iii) Furthermore the algorithm is able to handle low-structured domains (LSDs), which is important since the source model is specified in Declare - a language designed *especially* for modeling LSDs.

After choosing an appropriate algorithm a robust and domain-driven configuration is needed. A suggestion is shown in Tab. 1 (left). The *Dependency* parameter should be set to a value < 50.0 because the simulation step produces noise-free logs. It is therefore valid to assume that, according to this configuration, a path only observed once was also allowed in the source model and is therefore not negligible. The dependency value for such a single occurrence is 50. Consequently, there is no need for setting a *Relative-to-best* threshold higher than zero. If a dependency already has been accepted and the difference between the corresponding dependency value and a different dependency is lower than this threshold, this second dependency is also accepted. *All tasks connected* means that all non-initial tasks must have a predecessor and all non-final tasks must have a successor. The *Long distance dependencies* threshold is an additional threshold for identifying pairs of immediately or distant consecutive tasks. Setting this value to 100.0 means, at the example of tasks *A* and *B*, that *A* and *B* must be always consecutive and must have equal frequencies. The FHM requires some special attention for short loops like $\langle \dots AA \dots \rangle$ or $\langle \dots ABA \dots \rangle$. Setting both parameters to 0 means that if a task has been repeated at least once in one trace, we want to cover this behavior in the target model. Consequently we have set *Ignore loop dependency thresholds* to false. This configuration completes our tool chain for translating a Declare model to a trace-equivalent BPMN model.

4.6 Mining Declarative Process Models

Choosing an appropriate mining technique for discovering Declare models is much easier. The reason is that there are only two major approaches, one of whom is called *MINERful* [16] and the second, which is more a compilation of a mining technique and several pre- and post-processing steps, is called *Declare Maps Miner* (DMM) [21, 22]. We selected the latter bundle of techniques, where the decision this time is driven by a slight difference regarding quality tests [16] and our own experiences pertaining the tool integration. Though both approaches are comparable in terms of the result quality *MINERful* is a bit more sensitive to the configuration of two leading parameters, namely *confidence* and *interest factor*. However, *MINERful* outperforms the DMM in terms of computation time. But according to the experiences of the authors in [16], the latter is more appropriate in case of offline execution and is therefore also more appropriate for a highly automated tool chain. Finally the question of a target-aimed configuration is answered in Tab. 1 (right). Setting *Ignoring Event Types* to *false* is necessary since our source model is a BPMN model and therefore may allow for parallel execution of activities. A log is based on the linear time dimension, which means that we have to distinguish between the *start* and the *completion* of an activity, in order to represent a parallel execution. Since Declare does not allow for parallelism explicitly, we have to interpolate this behavior through considering the event types. Of course, this leads to a duplication of the tasks, compared to the original model. The threshold for the *Minimum Support* can be set to 100.0 because the log does not contain noise. The last parameter, called *Alpha* avoids that some considered rules are trivially true. This can be the case, for instance, with the *chainprecedence*(A, B) rule in Fig. 2 (d). If B is never executed, this rule is never violated and, therefore, trivially true. This would falsely consolidate this rule.

5 Evaluation

Within this section, we evaluate our approach in two stages. We do not provide an integrated implementation for our approach but describe a chain of suitable and well-established tools which are equipped with the desired functionalities.

5.1 Implementation

Many BPMN modeling tools provide simulation features, however, not all of them allow for the export of simulated traces. IYOPRO [23] allows for importing existing BPMN models. In order to run the simulation appropriately it is possible to influence the following basic parameters: (i) Inter-arrival times for *Start Events*, (ii) the duration of activities and (iii) probability distributions for the simulation of decisions at gateways. Additionally it is possible to modify the overall simulated execution time. These parameters influence the number and shape of generated traces and, therefore, it is not possible to determine this

parameter explicitly. In order to model the preferred trace length we have to run multiple simulations with different probability distributions for gateways. Using probability distributions already indicates that the passed paths through the process are computed randomly.

In contrast to BPMN there is only one simulation tool for Declare [20]. Since its primary application was the quality measurement of declarative process mining tools it is possible to specify the number of traces to generate as well as the maximum trace length explicitly. The Declare models are transformed into *Finite State Automata* and paths along them are chosen randomly. We export the traces in the XES standard format. For mining processes we use the well-known *ProM 6* toolkit [24]. For BPMN it provides the *BPMN Miner* extension [25], that contains, for instance, the FHM algorithm and for Declare we use the *DMM* plugin [15]. In both cases we first import the produced artificial event logs and afterwards choose and parameterize the mining algorithm. Additionally, we use ProM's conformance checking features for analyzing the transformation quality.

5.2 Used Evaluation Metrics

Since the final result is generated by process mining techniques we can reuse the corresponding well-known evaluation metrics. For reasons of comprehensibility we first give a small, fuzzy introduction to these metrics [13]:

- (1) *Fitness* (Recall): Which proportion of the logged traces can be parsed by the discovered model?
- (2) *Appropriateness* (Precision): What proportion of additional behavior is allowed by a model but is not present in the log?
- (3) *Generalization*: Discovered models should be able to parse unseen logs, too.
- (4) *Simplicity*: Discovered models should be as simple as possible. An important criterion is, for instance, the model size (number of nodes/edges).

For the evaluation we consider only the fitness and appropriateness. The generalization of the approach as well as simplicity of a model completely depends on the used process mining algorithm and cannot be controlled by the available simulation parameters. Furthermore, measuring these dimensions independently from the source model does not give any clue whether the model complexity is caused by inappropriate mining configuration or by the complexity of the source model. Since there are no comparable approaches so far, this paper focuses on checking the principal capability of this translation system in terms of correctness - which can be measured through the two metrics for fitness and appropriateness. For our calculations in the following subsection we use the formulae for fitness and appropriateness provided in [26].

5.3 Transformation Result Quality: Simple Models

In order to start measuring the transformation quality we first apply the introduced metrics to our small continuous examples shown in Fig. 2. The corresponding simulation configurations and measurement results are shown in Tab. 2. All

| N | L | Fitness | | | | Appropriateness | | | |
|-------|---|---------|-----|--------|--------|-----------------|-----|--------|-----|
| | | (a) | (b) | (c) | (d) | (a) | (b) | (c) | (d) |
| 10 | 3 | 1.0 | 1.0 | 0.7110 | 0.4932 | 1.0 | 1.0 | 0.9917 | 1.0 |
| 100 | 3 | 1.0 | 1.0 | 0.8911 | 0.6295 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1000 | 3 | 1.0 | 1.0 | 1.0 | 0.7286 | 1.0 | 1.0 | 1.0 | 1.0 |
| 10000 | 3 | 1.0 | 1.0 | 1.0 | 0.7286 | 1.0 | 1.0 | 1.0 | 1.0 |
| 10 | 6 | 1.0 | 1.0 | 0.713 | 0.6111 | 1.0 | 1.0 | 0.9929 | 1.0 |
| 100 | 6 | 1.0 | 1.0 | 0.9874 | 0.7257 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1000 | 6 | 1.0 | 1.0 | 1.0 | 0.9975 | 1.0 | 1.0 | 1.0 | 1.0 |
| 10000 | 6 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 10 | 9 | 1.0 | 1.0 | 0.713 | 0.6420 | 1.0 | 1.0 | 0.9929 | 1.0 |
| 100 | 9 | 1.0 | 1.0 | 1.0 | 0.7844 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1000 | 9 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 10000 | 9 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

Table 2: Quality: Models (a)-(d) shown in Fig. 2

measurements have been produced using the corresponding ProM replay plugins with anew generated 10000 sample traces for each of the four models. The experiments have been repeated ten times and the results have been averaged.

Though the used source model for this first evaluation are very simplistic, it is possible to discern four important facts. First, the two simplest models (cf. Fig. 2(a) and (b)) can be transformed correctly, as expected, with a very low amount of traces of short length. Secondly, the appropriateness is almost always 100%. The reason is that, the less traces are passed to the relevant process miner, the more restrictive is the resulting model. Both miners treat the traces as the only allowed behavior and, therefore, produce models that are as strict as the traces themselves. The third insight is that in the case of the more complex models (cf. Fig 2(c) and (d)) the fitness decreases. This means that for translating from BPMN to Declare more traces are required to raise the fitness, which is expected due to more execution alternatives. Finally, we have to point out that we are able to achieve 100% fitness and appropriateness because our simulation components generate noise-free logs.

5.4 Transformation Result Quality: Complex Models

Our second evaluation state is based on two more complex models than used in the previous subsection. The Declare source model is a model mined from real-life log data which was provided in the context of the *BPI Challenge 2014*⁵. Furthermore, the mined model has been used in [20] as evaluation data, too. The logs have been produced in the context of customer-service-desk interactions regarding disruptions of ICT services. Consequently, the model has been mined with the Declare Maps Miner ProM extension, too. The resulting model consists of 12 tasks and 15 constraints. Our more complex BPMN model has already been used in [27] in order to prove the understandability of BPMN models supported by human-readable textual work instructions. The model consists of 15

⁵ Log available at: <http://www.win.tue.nl/bpi/doku.php?id=2014:challenge>

| N | L | Fitness | App. |
|--------|----|---------|------|
| 100 | 24 | 0.6371 | 1.0 |
| 1000 | 24 | 0.8181 | 1.0 |
| 10000 | 24 | 0.9992 | 1.0 |
| 100000 | 24 | 1.0 | 1.0 |
| 100 | 36 | 0.6554 | 1.0 |
| 1000 | 36 | 0.8988 | 1.0 |
| 10000 | 36 | 0.9998 | 1.0 |
| 100000 | 36 | 1.0 | 1.0 |

| N | L | Fitness | App. |
|--------|----|---------|------|
| 10 | 15 | 0.5 | 1.0 |
| 100 | 15 | 0.6253 | 1.0 |
| 1000 | 15 | 0.7462 | 1.0 |
| 10000 | 15 | 0.8784 | 1.0 |
| 100000 | 36 | 0.9335 | 1.0 |

Table 3: Model translation quality BPI Ch. 2014 (l), Bread deliv. process (r)

tasks and allows for 48 different paths through a bread-delivery process. Again, we evaluated the translation quality with ten log files containing 10000 traces, respectively. The quality measurements, shown in Tab. 3, are averaged, too.

Both tables show that we were able to translate the models to a very high degree and confirm the findings of the previous evaluation step, which means that the quality is only a matter of fitness and, thus, target models produced with too few traces tend to be overfitted, which is an expected and well-known issue in machine learning. For these two example a significant higher amount of traces is required.

Additionally, we analyzed the performance of our approach only slightly, since it is based upon techniques that have already been analyzed regarding the computation time. Our evaluation has been performed on the following hardware: Dell Latitude E6430, intel Core i7 3720QM (4 x 2.6 GHz), 16 GB DDR3 internal memory, SSD, Windows 8 (64 bit). Translating models like our small continuous examples in Fig. 2 require only few traces and, therefore can be performed in an average time of one second. Translating our two more complex models require far more traces which leads to an average computation time of 8 (BPI Ch.) or 10 (Bread del.) seconds. For more precise and broader performance analysis, please consider the corresponding literature for the four used components [14,20,22,23].

6 Related Work

This paper relates to different streams of research: process modelling approaches and process model transformation. In general, the differences between declarative and imperative process modeling approaches have been discussed in [3] where both imperative and declarative languages are investigated with respect to process model understanding. The most relevant work in the context of process model transformation between these different paradigms is [8]. The paper describes an approach to derive imperative process models from declarative models. It utilizes a sequence of steps, leading from declarative constraints to regular expressions, then to a finite state automaton and finally to a petri net. To the best of our knowledge there is no other specific approach for the translation of

declarative process models. There are, however, different approaches that translate process models from one imperative language to another one, e.g., BPMN to BPEL [28]. Furthermore our work is related to the approach presented in [9]. There the main issue of writing cumbersome model transformation rules is solved providing a transformation approach that works on examples. Similarly our approach works on exemplary models but, in contrast, these are composed using simulation techniques which means that we prevent the user from any overhead. For our transformation approach we make use of different process model simulation [20] and process mining techniques [14,22] that have already been mentioned and described throughout the paper.

7 Conclusion and Future Work

The process model translation approach presented in this paper provides an alternative to classical model-to-model-translation based systems. It is based on the assumption that a representative event log can be used as a transfer medium in order to reduce the complexity of process model translation systems. In order to ensure suitability in practice we evaluated our approach on real-life data. Our evaluation showed that with a certain amount of simulated traces it is possible to cover the behavioral semantics of our exemplary source model in the log. However, in order to use the approach in real-life applications the general log expressiveness must be investigated in advance. The same applies to the expressiveness of the relevant pairs of process modeling languages. Furthermore there are pairs of languages where both provide a certain support for modeling relations beyond plain control-flow dependencies. In order to generate all relevant traces from large and highly branched BPMN models, it is necessary to find a more suitable algorithm to define appropriate probability distributions for decisions at gateways. This could be achieved, for instance, if the algorithm considers not only the number of outgoing edges of each gateway but also the branching factor of all subsequent gateways. These two improvements lead to a more accurate calculation of the maximum number of unique traces and, hence, of the number of required simulated traces.

References

1. S. Jablonski, “MOBILE: A modular workflow model and architecture,” in *Working Conference on Dynamic Modelling and Information Systems*, 1994.
2. W. van der Aalst, M. Pesic, and H. Schonenberg, “Declarative workflows: Balancing between flexibility and support,” *Computer Science - Research and Development*, vol. 23, no. 2, pp. 99–113, 2009.
3. P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, and H. Reijers, “Imperative versus declarative process modeling languages: An empirical investigation,” *BPM Workshops*, pp. 383–394, 2012.
4. R. Vaculín, R. Hull, T. Heath, C. Cochran, A. Nigam, and P. Sukaviriya, “Declarative business artifact centric modeling of decision and knowledge intensive business processes,” in *EDOC*, pp. 151–160, 2011.

5. M. Pesic and W. M. P. van der Aalst, "A declarative approach for flexible business processes management," in *BPM Workshops*, pp. 169–180, 2006.
6. T. Hildebrandt, R. R. Mukkamala, T. Slaats, and F. Zanitti, "Contracts for cross-organizational workflows as timed dynamic condition response graphs," *The Journal of Logic and Algebraic Programming*, vol. 82, no. 5, pp. 164–185, 2013.
7. M. Zeising, S. Schöning, and S. Jablonski, "Towards a Common Platform for the Support of Routine and Agile Business Processes," in *CollaborateCom*, 2014.
8. J. Prescher, C. Di Ciccio, and J. Mendling, "From Declarative Processes to Imperative Models," in *SIMPDA*, pp. 162–173, 2014.
9. M. Wimmer, M. Strommer, H. Kargl, and G. Kramler, "Towards Model Transformation Generation By-Example," *HICSS*, pp. 285–294, 2007.
10. D. Fahland, D. Lübke, J. Mendling, H. Reijers, B. Weber, M. Weidlich, and S. Zugal, "Declarative versus imperative process modeling languages: The issue of understandability," in *BPMDs*, pp. 353–366, 2009.
11. W. M. P. van der Aalst, "Business Process Simulation Revisited," *Enterprise and Organizational Modeling and Simulation*, vol. 63, pp. 1–14, 2010.
12. R. Stewart, *Simulation: the practice of model development and use*. John Wiley & Sons, 2004.
13. W. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, vol. 2. 2011.
14. A. Weijters and J. Ribeiro, "Flexible Heuristics Miner (FHM)," in *CIDM*, pp. 310–317, 2011.
15. F. Maggi, A. Mooij, and W. van der Aalst, "User-Guided Discovery of Declarative Process Models," in *CIDM*, 2011.
16. C. Di Ciccio and M. Mecella, "On the discovery of declarative control flows for artful processes," *TMIS*, vol. 5, no. 4, 2015.
17. S. Schöning, C. Cabanillas, S. Jablonski, and J. Mendling, "Mining the Organisational Perspective in Agile Business Processes," in *BPMDs*, vol. 214 of *LNBIP*, pp. 37–52, Springer, 2015.
18. W. M. P. Aalst, *Handbook on Business Process Management: Introduction, Methods, and Information Systems*. Springer Berlin Heidelberg, 2015.
19. J. Nakatumba, A. Rozinat, and N. Russell, "Business process simulation: How to get it right," in *International Handbook on BPM*, 2008.
20. "Generating Event Logs through the Simulation of Declare Models the Simulation of Declare Models," *EOMAS*, pp. 20–36, 2015.
21. F. M. Maggi, J. C. Bose, and W. van der Aalst, "Efficient Discovery of Understandable Declarative Process Models from Event Logs," in *Advanced Information Systems Engineering*, pp. 270–285, 2012.
22. F. M. Maggi, "Declarative Process Mining with the Declare Component of ProM," in *BPM (Demos)*, 2013.
23. E. Uhlmann, C. Gabriel, and N. Raue, "An automation approach based on workflows and software agents for industrial product-service systems," *CIRP*, pp. 341 – 346, 2015.
24. B. F. Dongen, A. K. A. Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. Aalst, ch. The ProM Framework: A New Era in Process Mining Tool Support, pp. 444–454. 2005.
25. R. Conforti, M. Dumas, L. García-Bañuelos, and M. La Rosa, "Bpmn miner: Automated discovery of bpmn process models with hierarchical structure," *Information Systems*, vol. 56, pp. 284–303, 2016.

26. W. Van der Aalst, A. Adriansyah, and B. van Dongen, “Replaying history on process models for conformance checking and performance analysis,” *Wiley Interdisciplinary Reviews: DM and KD*, vol. 2, no. 2, pp. 182–192, 2012.
27. R. Rodrigues, L. G. Azevedo, K. Revoredo, M. O. Barros, and H. Leopold, “BPME: An Experiment on Process Model Understandability Using Textual Work Instructions and BPMN Models,” in *SBES*, 2015.
28. J. C. Recker and J. Mendling, “On the translation between bpmn and bpel: Conceptual mismatch between process modeling languages,” in *CAISE Workshops*, pp. 521–532, 2006.